

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência
alternativa como acima descrito
repetição uma seqüência que pode se repetir.

Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 6
- 2: Y ← 40
- 3: GAMA ← 3
- 4: enquanto X < Y
- 5: X ← X + GAMA
- 6: se ((GAMA mod 3) ≠ 0)
- 7: GAMA ← GAMA + 1
- 8: fim{se}
- 9: GAMA ← GAMA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 3
- 2: B ← 23
- 3: enquanto A ≤ B
- 4: se ((B > 15) ∨ (A ≤ 9))
- 5: B ← B - 1
- 6: fim{se}
- 7: A ← A + 1
- 8: B ← B - 2
- 9: fim{enquanto}
- 10: A ← A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X ← 5
- 2: enquanto X < 10
- 3: Y ← 7
- 4: enquanto Y < 11
- 5: X ← X + 3
- 6: Y ← X + 2
- 7: fim{enquanto}
- 8: Y ← Y + 3
- 9: X ← X + 1
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 2
- 2: Y ← 5
- 3: se X < 4 ∧ Y > 4
- 4: X ← X - 3
- 5: senão
- 6: Y ← Y + 2
- 7: se X mod 2 ≠ 0
- 8: Y ← Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir.

Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  7
2: Y  $\leftarrow$  41
3: ALFA  $\leftarrow$  3
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALFA
6:   se ((ALFA mod 3)  $\neq$  0)
7:     ALFA  $\leftarrow$  ALFA + 1
8:   fim{se}
9:   ALFA  $\leftarrow$  ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  1
2: B  $\leftarrow$  21
3: enquanto A  $\leq$  B
4:   se ((B > 12)  $\vee$  (A  $\leq$  12))
5:     B  $\leftarrow$  B - 1
6:   fim{se}
7:   A  $\leftarrow$  A + 2
8:   B  $\leftarrow$  B - 3
9: fim{enquanto}
10: A  $\leftarrow$  A - 2
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  1
2: enquanto X < 11
3:   Y  $\leftarrow$  4
4:   enquanto Y < 14
5:     X  $\leftarrow$  X + 2
6:     Y  $\leftarrow$  X + 1
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 3
9:   X  $\leftarrow$  X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  1
2: Y  $\leftarrow$  5
3: se X < 6  $\wedge$  Y > 5
4:   X  $\leftarrow$  X - 3
5: senão
6:   Y  $\leftarrow$  Y + 1
7:   se X mod 3  $\neq$  0
8:     X  $\leftarrow$  Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N ≤ LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 9
- 2: Y \leftarrow 35
- 3: CTDS \leftarrow 3
- 4: enquanto X < Y
- 5: X \leftarrow X + CTDS
- 6: se ((CTDS mod 2) ≠ 0)
- 7: CTDS \leftarrow CTDS + 2
- 8: fim{se}
- 9: CTDS \leftarrow CTDS + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 3
- 2: B \leftarrow 21
- 3: enquanto A ≤ B
- 4: se ((B > 6) ∨ (A ≤ 14))
- 5: B \leftarrow B - 1
- 6: fim{se}
- 7: A \leftarrow A + 2
- 8: B \leftarrow B - 2
- 9: fim{enquanto}
- 10: A \leftarrow A - 2
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 5
- 2: enquanto X < 8
- 3: Y \leftarrow 2
- 4: enquanto Y < 13
- 5: X \leftarrow X + 2
- 6: Y \leftarrow X + 2
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 2
- 9: X \leftarrow X + 1
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 3
- 2: Y \leftarrow 1
- 3: se X < 5 ∧ Y > 4
- 4: X \leftarrow X - 1
- 5: senão
- 6: Y \leftarrow Y + 3
- 7: se X mod 3 ≠ 0
- 8: Y \leftarrow Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 7
- 2: Y ← 35
- 3: ALFA ← 2
- 4: enquanto X < Y
- 5: X ← X + ALFA
- 6: se ((ALFA mod 3) ≠ 0)
- 7: ALFA ← ALFA + 2
- 8: fim{se}
- 9: ALFA ← ALFA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 3
- 2: B ← 21
- 3: enquanto A ≤ B
- 4: se ((B > 12) ∨ (A ≤ 15))
- 5: B ← B - 2
- 6: fim{se}
- 7: A ← A + 2
- 8: B ← B - 2
- 9: fim{enquanto}
- 10: A ← A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X ← 5
- 2: enquanto X < 8
- 3: Y ← 4
- 4: enquanto Y < 12
- 5: X ← X + 2
- 6: Y ← X + 2
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 2
- 2: Y ← 9
- 3: se X < 6 ∧ Y > 3
- 4: X ← X - 1
- 5: senão
- 6: Y ← Y + 1
- 7: se X mod 3 ≠ 0
- 8: Y ← Y + 1
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato **se <condição> então bloco-A fim{se}** neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é **se <condição> então bloco-A senão bloco-B fim{se}** Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando **enquanto** que tem formato **enquanto <condição> bloco-A fim{enquanto}**. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando **fim{enquanto}** há um desvio incondicional para o início do bloco ou o **enquanto**.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  10
2: Y  $\leftarrow$  39
3: ALIN  $\leftarrow$  4
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALIN
6:   se ((ALIN mod 2)  $\neq$  0)
7:     ALIN  $\leftarrow$  ALIN + 1
8:   fim{se}
9:   ALIN  $\leftarrow$  ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  1
2: B  $\leftarrow$  22
3: enquanto A  $\leq$  B
4:   se ((B > 10)  $\vee$  (A  $\leq$  9))
5:     B  $\leftarrow$  B - 1
6:   fim{se}
7:   A  $\leftarrow$  A + 3
8:   B  $\leftarrow$  B - 2
9: fim{enquanto}
10: A  $\leftarrow$  A - 2
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  2
2: enquanto X < 13
3:   Y  $\leftarrow$  3
4:   enquanto Y < 13
5:     X  $\leftarrow$  X + 1
6:     Y  $\leftarrow$  X + 1
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 2
9:   X  $\leftarrow$  X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  5
2: Y  $\leftarrow$  2
3: se X < 6  $\wedge$  Y > 5
4:   X  $\leftarrow$  X - 3
5: senão
6:   Y  $\leftarrow$  Y + 1
7:   se X mod 3  $\neq$  0
8:     X  $\leftarrow$  Y + 2
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  7
2: Y  $\leftarrow$  41
3: ALIN  $\leftarrow$  3
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALIN
6:   se ((ALIN mod 2)  $\neq$  0)
7:     ALIN  $\leftarrow$  ALIN + 2
8:   fim{se}
9:   ALIN  $\leftarrow$  ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  2
2: B  $\leftarrow$  23
3: enquanto A  $\leq$  B
4:   se ((B > 7)  $\vee$  (A  $\leq$  8))
5:     B  $\leftarrow$  B - 1
6:   fim{se}
7:   A  $\leftarrow$  A + 1
8:   B  $\leftarrow$  B - 3
9: fim{enquanto}
10: A  $\leftarrow$  A - 3
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  3
2: enquanto X < 13
3:   Y  $\leftarrow$  2
4:   enquanto Y < 11
5:     X  $\leftarrow$  X + 1
6:     Y  $\leftarrow$  X + 1
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 1
9:   X  $\leftarrow$  X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  5
2: Y  $\leftarrow$  7
3: se X < 6  $\wedge$  Y > 3
4:   X  $\leftarrow$  X - 2
5: senão
6:   Y  $\leftarrow$  Y + 1
7:   se X mod 2  $\neq$  0
8:     Y  $\leftarrow$  Y + 3
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  8
2: Y  $\leftarrow$  39
3: ALFA  $\leftarrow$  3
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALFA
6:   se ((ALFA mod 3)  $\neq$  0)
7:     ALFA  $\leftarrow$  ALFA + 2
8:   fim{se}
9:   ALFA  $\leftarrow$  ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  1
2: B  $\leftarrow$  22
3: enquanto A  $\leq$  B
4:   se ((B > 11)  $\vee$  (A  $\leq$  6))
5:     B  $\leftarrow$  B - 1
6:   fim{se}
7:   A  $\leftarrow$  A + 2
8:   B  $\leftarrow$  B - 2
9: fim{enquanto}
10: A  $\leftarrow$  A - 3
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  3
2: enquanto X < 6
3:   Y  $\leftarrow$  6
4:   enquanto Y < 14
5:     X  $\leftarrow$  X + 3
6:     Y  $\leftarrow$  X + 1
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 2
9:   X  $\leftarrow$  X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  2
2: Y  $\leftarrow$  7
3: se X < 4  $\wedge$  Y > 4
4:   X  $\leftarrow$  X - 3
5: senão
6:   Y  $\leftarrow$  Y + 2
7:   se X mod 4  $\neq$  0
8:     X  $\leftarrow$  Y + 2
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9:   TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 7
2: Y ← 36
3: ALIN ← 3
4: enquanto X < Y
5:   X ← X + ALIN
6:   se ((ALIN mod 2) ≠ 0)
7:     ALIN ← ALIN + 1
8:   fim{se}
9:   ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 1
2: B ← 23
3: enquanto A ≤ B
4:   se ((B > 6) ∨ (A ≤ 13))
5:     B ← B - 1
6:   fim{se}
7:   A ← A + 1
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
1: X ← 3
2: enquanto X < 7
3:   Y ← 5
4:   enquanto Y < 11
5:     X ← X + 3
6:     Y ← X + 2
7:   fim{enquanto}
8:   Y ← Y + 1
9:   X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 2
2: Y ← 2
3: se X < 5 ∧ Y > 4
4:   X ← X - 3
5: senão
6:   Y ← Y + 3
7:   se X mod 4 ≠ 0
8:     X ← Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 6
- 2: Y ← 38
- 3: BETA ← 2
- 4: enquanto X < Y
- 5: X ← X + BETA
- 6: se ((BETA mod 3) ≠ 0)
- 7: BETA ← BETA + 1
- 8: fim{se}
- 9: BETA ← BETA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 2
- 2: B ← 21
- 3: enquanto A ≤ B
- 4: se ((B > 11) ∨ (A ≤ 10))
- 5: B ← B - 1
- 6: fim{se}
- 7: A ← A + 3
- 8: B ← B - 1
- 9: fim{enquanto}
- 10: A ← A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X ← 3
- 2: enquanto X < 11
- 3: Y ← 5
- 4: enquanto Y < 13
- 5: X ← X + 3
- 6: Y ← X + 1
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 6
- 2: Y ← 8
- 3: se X < 6 ∧ Y > 3
- 4: X ← X - 2
- 5: senão
- 6: Y ← Y + 2
- 7: se X mod 3 ≠ 0
- 8: X ← Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=sáida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  10
2: Y  $\leftarrow$  31
3: ALIN  $\leftarrow$  4
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALIN
6:   se ((ALIN mod 3)  $\neq$  0)
7:     ALIN  $\leftarrow$  ALIN + 1
8:   fim{se}
9:   ALIN  $\leftarrow$  ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  2
2: B  $\leftarrow$  22
3: enquanto A  $\leq$  B
4:   se ((B > 15)  $\vee$  (A  $\leq$  8))
5:     B  $\leftarrow$  B - 2
6:   fim{se}
7:   A  $\leftarrow$  A + 3
8:   B  $\leftarrow$  B - 2
9: fim{enquanto}
10: A  $\leftarrow$  A - 2
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  2
2: enquanto X < 12
3:   Y  $\leftarrow$  1
4:   enquanto Y < 12
5:     X  $\leftarrow$  X + 2
6:     Y  $\leftarrow$  X + 2
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 2
9:   X  $\leftarrow$  X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  4
2: Y  $\leftarrow$  3
3: se X < 6  $\wedge$  Y > 5
4:   X  $\leftarrow$  X - 1
5: senão
6:   Y  $\leftarrow$  Y + 1
7:   se X mod 3  $\neq$  0
8:     X  $\leftarrow$  Y + 3
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9: TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 9
2: Y ← 31
3: ALIN ← 4
4: enquanto X < Y
5:   X ← X + ALIN
6:   se ((ALIN mod 3) ≠ 0)
7:     ALIN ← ALIN + 2
8:   fim{se}
9:   ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 1
2: B ← 23
3: enquanto A ≤ B
4:   se ((B > 7) ∨ (A ≤ 13))
5:     B ← B - 2
6:   fim{se}
7:   A ← A + 1
8:   B ← B - 1
9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
1: X ← 1
2: enquanto X < 6
3:   Y ← 3
4:   enquanto Y < 15
5:     X ← X + 2
6:     Y ← X + 2
7:   fim{enquanto}
8:   Y ← Y + 3
9:   X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 2
2: Y ← 3
3: se X < 4 ∧ Y > 5
4:   X ← X - 3
5: senão
6:   Y ← Y + 2
7:   se X mod 2 ≠ 0
8:     Y ← Y + 2
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência
alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 10
- 2: Y ← 34
- 3: GAMA ← 3
- 4: enquanto X < Y
- 5: X ← X + GAMA
- 6: se ((GAMA mod 3) ≠ 0)
- 7: GAMA ← GAMA + 2
- 8: fim{se}
- 9: GAMA ← GAMA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 2
- 2: B ← 23
- 3: enquanto A ≤ B
- 4: se ((B > 10) ∨ (A ≤ 13))
- 5: B ← B - 2
- 6: fim{se}
- 7: A ← A + 2
- 8: B ← B - 3
- 9: fim{enquanto}
- 10: A ← A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X ← 2
- 2: enquanto X < 13
- 3: Y ← 7
- 4: enquanto Y < 11
- 5: X ← X + 2
- 6: Y ← X + 1
- 7: fim{enquanto}
- 8: Y ← Y + 3
- 9: X ← X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 7
- 2: Y ← 2
- 3: se X < 6 ∧ Y > 5
- 4: X ← X - 1
- 5: senão
- 6: Y ← Y + 1
- 7: se X mod 2 ≠ 0
- 8: Y ← Y + 3
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N ≤ LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 10
- 2: Y \leftarrow 34
- 3: BETA \leftarrow 2
- 4: enquanto X < Y
- 5: X \leftarrow X + BETA
- 6: se ((BETA mod 2) ≠ 0)
- 7: BETA \leftarrow BETA + 2
- 8: fim{se}
- 9: BETA \leftarrow BETA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 3
- 2: B \leftarrow 22
- 3: enquanto A ≤ B
- 4: se ((B > 8) ∨ (A ≤ 14))
- 5: B \leftarrow B - 3
- 6: fim{se}
- 7: A \leftarrow A + 1
- 8: B \leftarrow B - 3
- 9: fim{enquanto}
- 10: A \leftarrow A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 4
- 2: enquanto X < 10
- 3: Y \leftarrow 5
- 4: enquanto Y < 12
- 5: X \leftarrow X + 2
- 6: Y \leftarrow X + 1
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 3
- 9: X \leftarrow X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 4
- 2: Y \leftarrow 8
- 3: se X < 6 ∧ Y > 4
- 4: X \leftarrow X - 2
- 5: senão
- 6: Y \leftarrow Y + 1
- 7: se X mod 3 ≠ 0
- 8: X \leftarrow Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência **alternativa** como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 9
- 2: Y ← 40
- 3: ALFA ← 4
- 4: enquanto X < Y
- 5: X ← X + ALFA
- 6: se ((ALFA mod 3) ≠ 0)
- 7: ALFA ← ALFA + 1
- 8: fim{se}
- 9: ALFA ← ALFA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 2
- 2: B ← 23
- 3: enquanto A ≤ B
- 4: se ((B > 9) ∨ (A ≤ 10))
- 5: B ← B - 1
- 6: fim{se}
- 7: A ← A + 2
- 8: B ← B - 3
- 9: fim{enquanto}
- 10: A ← A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X ← 5
- 2: enquanto X < 7
- 3: Y ← 4
- 4: enquanto Y < 15
- 5: X ← X + 3
- 6: Y ← X + 1
- 7: fim{enquanto}
- 8: Y ← Y + 3
- 9: X ← X + 1
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 7
- 2: Y ← 5
- 3: se X < 5 ∧ Y > 3
- 4: X ← X - 2
- 5: senão
- 6: Y ← Y + 3
- 7: se X mod 2 ≠ 0
- 8: Y ← Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=sáida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprende lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência
alternativa como acima descrito
repetição uma sequência que pode se repetir.

Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que os programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação,

pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

* um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
* um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato **se <condição> então bloco-A fim{se}** neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é **se <condição> então bloco-A senão bloco-B fim{se}** Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando **enquanto** que tem formato **enquanto <condição> bloco-A fim{enquanto}**. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando **fim{enquanto}** há um desvio incondicional para o início do bloco ou o **enquanto**.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9:   TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*),

ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

Para você fazer

Ex. 1

```
1: X ← 7
2: Y ← 41
3: CTDS ← 2
4: enquanto X < Y
5:   X ← X + CTDS
6:   se ((CTDS mod 3) ≠ 0)
7:     CTDS ← CTDS + 1
8:   fim{se}
9:   CTDS ← CTDS + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 3
2: B ← 21
3: enquanto A ≤ B
4:   se ((B > 10) ∨ (A ≤ 11))
5:     B ← B - 1
6:   fim{se}
7:   A ← A + 3
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 1
2: enquanto X < 8
3:   Y ← 3
4:   enquanto Y < 14
5:     X ← X + 3
6:     Y ← X + 1
7:   fim{enquanto}
8:   Y ← Y + 3
9:   X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 9
2: Y ← 2
3: se X < 5 ∧ Y > 3
4:   X ← X - 1
5: senão
6:   Y ← Y + 3
7:   se X mod 3 ≠ 0
8:     Y ← Y + 2
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a ideia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR $<$ X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar ideias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N \leq LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 10
- 2: Y \leftarrow 42
- 3: ALIN \leftarrow 4
- 4: enquanto X $<$ Y
- 5: X \leftarrow X + ALIN
- 6: se ((ALIN mod 3) \neq 0)
- 7: ALIN \leftarrow ALIN + 2
- 8: fim{se}
- 9: ALIN \leftarrow ALIN + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 1
- 2: B \leftarrow 21
- 3: enquanto A \leq B
- 4: se ((B $>$ 6) \vee (A \leq 13))
- 5: B \leftarrow B - 1
- 6: fim{se}
- 7: A \leftarrow A + 1
- 8: B \leftarrow B - 3
- 9: fim{enquanto}
- 10: A \leftarrow A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 3
- 2: enquanto X $<$ 6
- 3: Y \leftarrow 6
- 4: enquanto Y $<$ 14
- 5: X \leftarrow X + 2
- 6: Y \leftarrow X + 1
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 2
- 9: X \leftarrow X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 3
- 2: Y \leftarrow 9
- 3: se X $<$ 6 \wedge Y $>$ 4
- 4: X \leftarrow X - 2
- 5: senão
- 6: Y \leftarrow Y + 2
- 7: se X mod 2 \neq 0
- 8: Y \leftarrow Y + 1
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9:   TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 8
2: Y ← 32
3: GAMA ← 2
4: enquanto X < Y
5:   X ← X + GAMA
6:   se ((GAMA mod 3) ≠ 0)
7:     GAMA ← GAMA + 2
8:   fim{se}
9:   GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 2
2: B ← 22
3: enquanto A ≤ B
4:   se ((B > 6) ∨ (A ≤ 11))
5:     B ← B - 3
6:   fim{se}
7:   A ← A + 3
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 5
2: enquanto X < 12
3:   Y ← 6
4:   enquanto Y < 13
5:     X ← X + 2
6:     Y ← X + 1
7:   fim{enquanto}
8:   Y ← Y + 1
9:   X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 4
2: Y ← 3
3: se X < 6 ∧ Y > 5
4:   X ← X - 3
5: senão
6:   Y ← Y + 2
7:   se X mod 3 ≠ 0
8:     X ← Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  7
2: Y  $\leftarrow$  41
3: CTDS  $\leftarrow$  3
4: enquanto X < Y
5:   X  $\leftarrow$  X + CTDS
6:   se ((CTDS mod 3)  $\neq$  0)
7:     CTDS  $\leftarrow$  CTDS + 2
8:   fim{se}
9:   CTDS  $\leftarrow$  CTDS + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  1
2: B  $\leftarrow$  21
3: enquanto A  $\leq$  B
4:   se ((B > 6)  $\vee$  (A  $\leq$  10))
5:     B  $\leftarrow$  B - 3
6:   fim{se}
7:   A  $\leftarrow$  A + 3
8:   B  $\leftarrow$  B - 2
9: fim{enquanto}
10: A  $\leftarrow$  A - 2
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  4
2: enquanto X < 11
3:   Y  $\leftarrow$  2
4:   enquanto Y < 11
5:     X  $\leftarrow$  X + 2
6:     Y  $\leftarrow$  X + 1
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 1
9:   X  $\leftarrow$  X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  4
2: Y  $\leftarrow$  4
3: se X < 4  $\wedge$  Y > 5
4:   X  $\leftarrow$  X - 1
5: senão
6:   Y  $\leftarrow$  Y + 2
7:   se X mod 4  $\neq$  0
8:     X  $\leftarrow$  Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: vá para a instrução x : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato se <condição> então bloco-A fim{se} neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é se <condição> então bloco-A senão bloco-B fim{se} Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando enquanto que tem formato enquanto <condição> bloco-A fim{enquanto}. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando fim{enquanto} há um desvio incondicional para o início do bloco ou o enquanto.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 10
- 2: Y ← 37
- 3: GAMA ← 4
- 4: enquanto X < Y
- 5: X ← X + GAMA
- 6: se ((GAMA mod 2) ≠ 0)
- 7: GAMA ← GAMA + 1
- 8: fim{se}
- 9: GAMA ← GAMA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 3
- 2: B ← 21
- 3: enquanto A ≤ B
- 4: se ((B > 14) ∨ (A ≤ 11))
- 5: B ← B - 1
- 6: fim{se}
- 7: A ← A + 2
- 8: B ← B - 2
- 9: fim{enquanto}
- 10: A ← A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X ← 5
- 2: enquanto X < 6
- 3: Y ← 2
- 4: enquanto Y < 14
- 5: X ← X + 2
- 6: Y ← X + 2
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 3
- 2: Y ← 3
- 3: se X < 5 ∧ Y > 3
- 4: X ← X - 1
- 5: senão
- 6: Y ← Y + 1
- 7: se X mod 2 ≠ 0
- 8: Y ← Y + 3
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 8
- 2: Y ← 32
- 3: GAMA ← 3
- 4: enquanto X < Y
- 5: X ← X + GAMA
- 6: se ((GAMA mod 3) ≠ 0)
- 7: GAMA ← GAMA + 2
- 8: fim{se}
- 9: GAMA ← GAMA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 3
- 2: B ← 22
- 3: enquanto A ≤ B
- 4: se ((B > 14) ∨ (A ≤ 9))
- 5: B ← B - 2
- 6: fim{se}
- 7: A ← A + 3
- 8: B ← B - 2
- 9: fim{enquanto}
- 10: A ← A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X ← 4
- 2: enquanto X < 8
- 3: Y ← 2
- 4: enquanto Y < 14
- 5: X ← X + 1
- 6: Y ← X + 1
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 3
- 2: Y ← 3
- 3: se X < 5 ∧ Y > 3
- 4: X ← X - 1
- 5: senão
- 6: Y ← Y + 2
- 7: se X mod 4 ≠ 0
- 8: Y ← Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N \leq LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 6
- 2: Y \leftarrow 40
- 3: ALFA \leftarrow 2
- 4: enquanto X < Y
- 5: X \leftarrow X + ALFA
- 6: se ((ALFA mod 3) \neq 0)
- 7: ALFA \leftarrow ALFA + 1
- 8: fim{se}
- 9: ALFA \leftarrow ALFA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 3
- 2: B \leftarrow 23
- 3: enquanto A \leq B
- 4: se ((B > 12) \vee (A \leq 13))
- 5: B \leftarrow B - 2
- 6: fim{se}
- 7: A \leftarrow A + 3
- 8: B \leftarrow B - 3
- 9: fim{enquanto}
- 10: A \leftarrow A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 3
- 2: enquanto X < 10
- 3: Y \leftarrow 6
- 4: enquanto Y < 13
- 5: X \leftarrow X + 2
- 6: Y \leftarrow X + 1
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 1
- 9: X \leftarrow X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 3
- 2: Y \leftarrow 3
- 3: se X < 5 \wedge Y > 4
- 4: X \leftarrow X - 1
- 5: senão
- 6: Y \leftarrow Y + 3
- 7: se X mod 4 \neq 0
- 8: X \leftarrow Y + 3
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a ideia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar ideias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  8
2: Y  $\leftarrow$  33
3: ALIN  $\leftarrow$  4
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALIN
6:   se ((ALIN mod 2)  $\neq$  0)
7:     ALIN  $\leftarrow$  ALIN + 2
8:   fim{se}
9:   ALIN  $\leftarrow$  ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  2
2: B  $\leftarrow$  23
3: enquanto A  $\leq$  B
4:   se ((B > 7)  $\vee$  (A  $\leq$  13))
5:     B  $\leftarrow$  B - 3
6:   fim{se}
7:   A  $\leftarrow$  A + 2
8:   B  $\leftarrow$  B - 2
9: fim{enquanto}
10: A  $\leftarrow$  A - 3
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  5
2: enquanto X < 11
3:   Y  $\leftarrow$  4
4:   enquanto Y < 11
5:     X  $\leftarrow$  X + 1
6:     Y  $\leftarrow$  X + 2
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 3
9:   X  $\leftarrow$  X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  3
2: Y  $\leftarrow$  3
3: se X < 4  $\wedge$  Y > 4
4:   X  $\leftarrow$  X - 1
5: senão
6:   Y  $\leftarrow$  Y + 3
7:   se X mod 3  $\neq$  0
8:     X  $\leftarrow$  Y + 2
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python. Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR $<$ X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N \leq LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 9
- 2: Y \leftarrow 42
- 3: CTDS \leftarrow 3
- 4: enquanto X $<$ Y
- 5: X \leftarrow X + CTDS
- 6: se ((CTDS mod 3) \neq 0)
- 7: CTDS \leftarrow CTDS + 1
- 8: fim{se}
- 9: CTDS \leftarrow CTDS + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 3
- 2: B \leftarrow 21
- 3: enquanto A \leq B
- 4: se ((B $>$ 9) \vee (A \leq 12))
- 5: B \leftarrow B - 3
- 6: fim{se}
- 7: A \leftarrow A + 1
- 8: B \leftarrow B - 3
- 9: fim{enquanto}
- 10: A \leftarrow A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 2
- 2: enquanto X $<$ 10
- 3: Y \leftarrow 6
- 4: enquanto Y $<$ 13
- 5: X \leftarrow X + 3
- 6: Y \leftarrow X + 2
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 2
- 9: X \leftarrow X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 9
- 2: Y \leftarrow 3
- 3: se X $<$ 6 \wedge Y $>$ 3
- 4: X \leftarrow X - 1
- 5: senão
- 6: Y \leftarrow Y + 2
- 7: se X mod 3 \neq 0
- 8: Y \leftarrow Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9:   TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 8
2: Y ← 41
3: GAMA ← 3
4: enquanto X < Y
5:   X ← X + GAMA
6:   se ((GAMA mod 3) ≠ 0)
7:     GAMA ← GAMA + 2
8:   fim{se}
9:   GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 1
2: B ← 23
3: enquanto A ≤ B
4:   se ((B > 12) ∨ (A ≤ 13))
5:     B ← B - 2
6:   fim{se}
7:   A ← A + 2
8:   B ← B - 3
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 4
2: enquanto X < 6
3:   Y ← 4
4:   enquanto Y < 15
5:     X ← X + 1
6:     Y ← X + 1
7:   fim{enquanto}
8:   Y ← Y + 2
9:   X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 9
2: Y ← 1
3: se X < 5 ∧ Y > 4
4:   X ← X - 3
5: senão
6:   Y ← Y + 1
7:   se X mod 2 ≠ 0
8:     X ← Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência **alternativa** como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N ≤ LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 10
- 2: Y \leftarrow 42
- 3: ALIN \leftarrow 3
- 4: enquanto X < Y
- 5: X \leftarrow X + ALIN
- 6: se ((ALIN mod 2) ≠ 0)
- 7: ALIN \leftarrow ALIN + 2
- 8: fim{se}
- 9: ALIN \leftarrow ALIN + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 1
- 2: B \leftarrow 23
- 3: enquanto A ≤ B
- 4: se ((B > 8) ∨ (A ≤ 15))
- 5: B \leftarrow B - 1
- 6: fim{se}
- 7: A \leftarrow A + 1
- 8: B \leftarrow B - 2
- 9: fim{enquanto}
- 10: A \leftarrow A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 2
- 2: enquanto X < 11
- 3: Y \leftarrow 2
- 4: enquanto Y < 13
- 5: X \leftarrow X + 2
- 6: Y \leftarrow X + 1
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 3
- 9: X \leftarrow X + 1
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 3
- 2: Y \leftarrow 7
- 3: se X < 5 ∧ Y > 3
- 4: X \leftarrow X - 1
- 5: senão
- 6: Y \leftarrow Y + 2
- 7: se X mod 4 ≠ 0
- 8: X \leftarrow Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 6
- 2: Y ← 33
- 3: CTDS ← 4
- 4: enquanto X < Y
- 5: X ← X + CTDS
- 6: se ((CTDS mod 2) ≠ 0)
- 7: CTDS ← CTDS + 1
- 8: fim{se}
- 9: CTDS ← CTDS + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 1
- 2: B ← 22
- 3: enquanto A ≤ B
- 4: se ((B > 8) ∨ (A ≤ 9))
- 5: B ← B - 1
- 6: fim{se}
- 7: A ← A + 2
- 8: B ← B - 3
- 9: fim{enquanto}
- 10: A ← A - 1
- 11: escreva (A + B)

Ex. 3

- 1: X ← 5
- 2: enquanto X < 10
- 3: Y ← 6
- 4: enquanto Y < 14
- 5: X ← X + 1
- 6: Y ← X + 1
- 7: fim{enquanto}
- 8: Y ← Y + 3
- 9: X ← X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 5
- 2: Y ← 9
- 3: se X < 6 ∧ Y > 4
- 4: X ← X - 3
- 5: senão
- 6: Y ← Y + 2
- 7: se X mod 2 ≠ 0
- 8: Y ← Y + 1
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saiada de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES \leftarrow 0
- 4: TESTADOR \leftarrow 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES \leftarrow DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR \leftarrow TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A \leftarrow 1; B \leftarrow 1; N \leftarrow 3
- 3: enquanto N ≤ LIMITE
- 4: R \leftarrow A + B
- 5: A \leftarrow B
- 6: B \leftarrow R
- 7: N \leftarrow N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X \leftarrow 9
- 2: Y \leftarrow 36
- 3: BETA \leftarrow 4
- 4: enquanto X < Y
- 5: X \leftarrow X + BETA
- 6: se ((BETA mod 3) ≠ 0)
- 7: BETA \leftarrow BETA + 2
- 8: fim{se}
- 9: BETA \leftarrow BETA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A \leftarrow 1
- 2: B \leftarrow 22
- 3: enquanto A ≤ B
- 4: se ((B > 7) ∨ (A ≤ 8))
- 5: B \leftarrow B - 1
- 6: fim{se}
- 7: A \leftarrow A + 3
- 8: B \leftarrow B - 2
- 9: fim{enquanto}
- 10: A \leftarrow A - 2
- 11: escreva (A + B)

Ex. 3

- 1: X \leftarrow 4
- 2: enquanto X < 10
- 3: Y \leftarrow 4
- 4: enquanto Y < 11
- 5: X \leftarrow X + 1
- 6: Y \leftarrow X + 2
- 7: fim{enquanto}
- 8: Y \leftarrow Y + 3
- 9: X \leftarrow X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X \leftarrow 8
- 2: Y \leftarrow 7
- 3: se X < 6 ∧ Y > 4
- 4: X \leftarrow X - 2
- 5: senão
- 6: Y \leftarrow Y + 3
- 7: se X mod 2 ≠ 0
- 8: Y \leftarrow Y + 3
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9: TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 7
2: Y ← 33
3: GAMA ← 4
4: enquanto X < Y
5:   X ← X + GAMA
6:   se ((GAMA mod 2) ≠ 0)
7:     GAMA ← GAMA + 2
8:   fim{se}
9:   GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 2
2: B ← 23
3: enquanto A ≤ B
4:   se ((B > 15) ∨ (A ≤ 11))
5:     B ← B - 3
6:   fim{se}
7:   A ← A + 1
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 3
2: enquanto X < 7
3:   Y ← 5
4:   enquanto Y < 13
5:     X ← X + 2
6:     Y ← X + 2
7:   fim{enquanto}
8:   Y ← Y + 2
9:   X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 5
2: Y ← 5
3: se X < 6 ∧ Y > 5
4:   X ← X - 2
5: senão
6:   Y ← Y + 3
7:   se X mod 3 ≠ 0
8:     X ← Y + 3
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência
alternativa como acima descrito
repetição uma seqüência que pode se repetir.

Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9: TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 10
2: Y ← 41
3: GAMA ← 4
4: enquanto X < Y
5:   X ← X + GAMA
6:   se ((GAMA mod 3) ≠ 0)
7:     GAMA ← GAMA + 1
8:   fim{se}
9:   GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 2
2: B ← 23
3: enquanto A ≤ B
4:   se ((B > 6) ∨ (A ≤ 6))
5:     B ← B - 1
6:   fim{se}
7:   A ← A + 1
8:   B ← B - 1
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 4
2: enquanto X < 10
3:   Y ← 6
4:   enquanto Y < 14
5:     X ← X + 3
6:     Y ← X + 1
7:   fim{enquanto}
8:   Y ← Y + 2
9:   X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 5
2: Y ← 1
3: se X < 6 ∧ Y > 5
4:   X ← X - 1
5: senão
6:   Y ← Y + 3
7:   se X mod 2 ≠ 0
8:     Y ← Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 8
- 2: Y ← 38
- 3: ALFA ← 3
- 4: enquanto X < Y
- 5: X ← X + ALFA
- 6: se ((ALFA mod 3) ≠ 0)
- 7: ALFA ← ALFA + 1
- 8: fim{se}
- 9: ALFA ← ALFA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 3
- 2: B ← 21
- 3: enquanto A ≤ B
- 4: se ((B > 13) ∨ (A ≤ 8))
- 5: B ← B - 2
- 6: fim{se}
- 7: A ← A + 3
- 8: B ← B - 2
- 9: fim{enquanto}
- 10: A ← A - 3
- 11: escreva (A + B)

Ex. 3

- 1: X ← 3
- 2: enquanto X < 9
- 3: Y ← 1
- 4: enquanto Y < 12
- 5: X ← X + 3
- 6: Y ← X + 2
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 2
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 8
- 2: Y ← 3
- 3: se X < 5 ∧ Y > 3
- 4: X ← X - 2
- 5: senão
- 6: Y ← Y + 1
- 7: se X mod 3 ≠ 0
- 8: Y ← Y + 2
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  10
2: Y  $\leftarrow$  34
3: ALFA  $\leftarrow$  4
4: enquanto X < Y
5:   X  $\leftarrow$  X + ALFA
6:   se ((ALFA mod 2)  $\neq$  0)
7:     ALFA  $\leftarrow$  ALFA + 1
8:   fim{se}
9:   ALFA  $\leftarrow$  ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  2
2: B  $\leftarrow$  22
3: enquanto A  $\leq$  B
4:   se ((B > 11)  $\vee$  (A  $\leq$  11))
5:     B  $\leftarrow$  B - 1
6:   fim{se}
7:   A  $\leftarrow$  A + 3
8:   B  $\leftarrow$  B - 1
9: fim{enquanto}
10: A  $\leftarrow$  A - 3
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  5
2: enquanto X < 11
3:   Y  $\leftarrow$  6
4:   enquanto Y < 13
5:     X  $\leftarrow$  X + 3
6:     Y  $\leftarrow$  X + 1
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 2
9:   X  $\leftarrow$  X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  7
2: Y  $\leftarrow$  2
3: se X < 4  $\wedge$  Y > 3
4:   X  $\leftarrow$  X - 2
5: senão
6:   Y  $\leftarrow$  Y + 2
7:   se X mod 3  $\neq$  0
8:     X  $\leftarrow$  Y + 3
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 7
- 2: Y ← 39
- 3: GAMA ← 4
- 4: enquanto X < Y
- 5: X ← X + GAMA
- 6: se ((GAMA mod 3) ≠ 0)
- 7: GAMA ← GAMA + 2
- 8: fim{se}
- 9: GAMA ← GAMA + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 2
- 2: B ← 22
- 3: enquanto A ≤ B
- 4: se ((B > 15) ∨ (A ≤ 7))
- 5: B ← B - 3
- 6: fim{se}
- 7: A ← A + 3
- 8: B ← B - 1
- 9: fim{enquanto}
- 10: A ← A - 2
- 11: escreva (A + B)

Ex. 3

- 1: X ← 1
- 2: enquanto X < 6
- 3: Y ← 2
- 4: enquanto Y < 13
- 5: X ← X + 3
- 6: Y ← X + 2
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 4
- 2: Y ← 2
- 3: se X < 5 ∧ Y > 3
- 4: X ← X - 2
- 5: senão
- 6: Y ← Y + 3
- 7: se X mod 4 ≠ 0
- 8: Y ← Y + 1
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saição de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

seqüência 2 ou mais comandos em seqüência

alternativa como acima descrito

repetição uma seqüência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.
Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por ←, então, por exemplo *A ← 0* significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendizado nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

- 1: algoritmo PRIMO
- 2: leia(X)
- 3: DIVISORES ← 0
- 4: TESTADOR ← 2
- 5: enquanto TESTADOR < X
- 6: se X MOD TESTADOR = 0
- 7: DIVISORES ← DIVISORES + 1
- 8: fim{se}
- 9: TESTADOR ← TESTADOR + 1
- 10: fim{enquanto}
- 11: se DIVISORES = 0
- 12: escreva("PRIMO")
- 13: senão
- 14: escreva("NÃO PRIMO")
- 15: fim{se}

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a seqüência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A seqüência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

- 1: leia LIMITE
- 2: A ← 1; B ← 1; N ← 3
- 3: enquanto N ≤ LIMITE
- 4: R ← A + B
- 5: A ← B
- 6: B ← R
- 7: N ← N + 1
- 8: fim{enquanto}
- 9: escreva R

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

- 1: X ← 10
- 2: Y ← 33
- 3: ALIN ← 4
- 4: enquanto X < Y
- 5: X ← X + ALIN
- 6: se ((ALIN mod 3) ≠ 0)
- 7: ALIN ← ALIN + 1
- 8: fim{se}
- 9: ALIN ← ALIN + 1
- 10: fim{enquanto}
- 11: escreva X

Ex. 2

- 1: A ← 3
- 2: B ← 22
- 3: enquanto A ≤ B
- 4: se ((B > 11) ∨ (A ≤ 10))
- 5: B ← B - 3
- 6: fim{se}
- 7: A ← A + 2
- 8: B ← B - 2
- 9: fim{enquanto}
- 10: A ← A - 2
- 11: escreva (A + B)

Ex. 3

- 1: X ← 4
- 2: enquanto X < 12
- 3: Y ← 3
- 4: enquanto Y < 13
- 5: X ← X + 3
- 6: Y ← X + 2
- 7: fim{enquanto}
- 8: Y ← Y + 1
- 9: X ← X + 3
- 10: fim{enquanto}
- 11: escreva (X-Y)

Ex. 4

- 1: X ← 8
- 2: Y ← 1
- 3: se X < 6 ∧ Y > 5
- 4: X ← X - 3
- 5: senão
- 6: Y ← Y + 2
- 7: se X mod 4 ≠ 0
- 8: Y ← Y + 3
- 9: fim{se}
- 10: fim{se}
- 11: escreva X+Y

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*. Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais (>, <, ≤, ≥, =, ≠, ∈, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (∧) e OU (∨) além de NÃO (¬). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES ← 0
4: TESTADOR ← 2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES ← DIVISORES + 1
8:   fim{se}
9: TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:   R ← A + B
5:   A ← B
6:   B ← R
7:   N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X ← 8
2: Y ← 34
3: ALIN ← 3
4: enquanto X < Y
5:   X ← X + ALIN
6:   se ((ALIN mod 2) ≠ 0)
7:     ALIN ← ALIN + 2
8:   fim{se}
9:   ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A ← 1
2: B ← 22
3: enquanto A ≤ B
4:   se ((B > 13) ∨ (A ≤ 14))
5:     B ← B - 3
6:   fim{se}
7:   A ← A + 2
8:   B ← B - 2
9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
1: X ← 1
2: enquanto X < 6
3:   Y ← 5
4:   enquanto Y < 15
5:     X ← X + 3
6:     Y ← X + 2
7:   fim{enquanto}
8:   Y ← Y + 3
9:   X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X ← 8
2: Y ← 1
3: se X < 6 ∧ Y > 5
4:   X ← X - 1
5: senão
6:   Y ← Y + 3
7:   se X mod 2 ≠ 0
8:     Y ← Y + 3
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4



Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em meados da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio. **E/S** Este tipo de instrução responde pela E=entrada e S=saida de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva - prioridades e parêntesis, etc etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre si.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x*: Neste caso a seguir será executada a instrução *x* e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spagueti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Este conceito já havia sido posto por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de

máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

- * um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
- * um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a idéia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. A poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por \leftarrow , então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável *A*.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, *: multiplicação, -: subtração, /: divisão real, //: divisão inteira, MOD: resto de, **: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, \leq , \geq , $=$, \neq , \in , ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E (\wedge) e OU (\vee) além de NÃO (\sim). O comando alternativo tem o formato *se <condição> então bloco-A fim{se}* neste comando o bloco de comandos *A* só será executado se a condição citada for verdadeira. Um formato alternativo é *se <condição> então bloco-A senão bloco-B fim{se}* Agora, se a condição é verdadeira, *A* é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando *enquanto* que tem formato *enquanto <condição> bloco-A fim{enquanto}*. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando *fim{enquanto}* há um desvio incondicional para o início do bloco ou o *enquanto*.

Um exemplo Vamos escrever um algoritmo para testar se o número *X* é primo ou não.

```
1: algoritmo PRIMO
2: leia(X)
3: DIVISORES  $\leftarrow$  0
4: TESTADOR  $\leftarrow$  2
5: enquanto TESTADOR < X
6:   se X MOD TESTADOR = 0
7:     DIVISORES  $\leftarrow$  DIVISORES + 1
8:   fim{se}
9:   TESTADOR  $\leftarrow$  TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:   escreva("PRIMO")
13: senão
14:   escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão

com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na sintaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número *n* e imprima o *n*-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A  $\leftarrow$  1; B  $\leftarrow$  1; N  $\leftarrow$  3
3: enquanto N  $\leq$  LIMITE
4:   R  $\leftarrow$  A + B
5:   A  $\leftarrow$  B
6:   B  $\leftarrow$  R
7:   N  $\leftarrow$  N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de sintaxe da linguagem:

☞ Para você fazer

Ex. 1

```
1: X  $\leftarrow$  10
2: Y  $\leftarrow$  42
3: CTDS  $\leftarrow$  4
4: enquanto X < Y
5:   X  $\leftarrow$  X + CTDS
6:   se ((CTDS mod 2)  $\neq$  0)
7:     CTDS  $\leftarrow$  CTDS + 2
8:   fim{se}
9:   CTDS  $\leftarrow$  CTDS + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
1: A  $\leftarrow$  1
2: B  $\leftarrow$  21
3: enquanto A  $\leq$  B
4:   se ((B > 8)  $\vee$  (A  $\leq$  10))
5:     B  $\leftarrow$  B - 1
6:   fim{se}
7:   A  $\leftarrow$  A + 2
8:   B  $\leftarrow$  B - 1
9: fim{enquanto}
10: A  $\leftarrow$  A - 3
11: escreva (A + B)
```

Ex. 3

```
1: X  $\leftarrow$  4
2: enquanto X < 7
3:   Y  $\leftarrow$  7
4:   enquanto Y < 14
5:     X  $\leftarrow$  X + 3
6:     Y  $\leftarrow$  X + 2
7:   fim{enquanto}
8:   Y  $\leftarrow$  Y + 2
9:   X  $\leftarrow$  X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
1: X  $\leftarrow$  8
2: Y  $\leftarrow$  4
3: se X < 6  $\wedge$  Y > 4
4:   X  $\leftarrow$  X - 2
5: senão
6:   Y  $\leftarrow$  Y + 2
7:   se X mod 4  $\neq$  0
8:     X  $\leftarrow$  Y + 1
9:   fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4
---	---	---	---

